Mixed Discrete and Continuous Algorithms for Scheduling Airborne Astronomy Observations

Jeremy Frank and Elif Kürklü*
{frank,ekurklu}@email.arc.nasa.gov

NASA Ames Research Center
Mail Stop N269-3
Moffett Field CA 94035-1000

Abstract. We describe the problem of scheduling astronomy observations for the Stratospheric Observatory for Infrared Astronomy, an airborne telescope. The problem requires maximizing the number of requested observations scheduled subject to a mixture of discrete and continuous constraints relating the feasibility of an astronomical observation to the position and time at which the observation begins, telescope elevation limits, Special Use Airspace limitations, and available fuel. Solving the problem requires making discrete choices (e.g. selection and sequencing of observations) and continuous ones (e.g. takeoff time and setup actions for observations by repositioning the aircraft). Previously, we developed an algorithm called ForwardPlanner using a combination of AI and OR techniques including progression planning, lookahead heuristics, stochastic sampling and numerical optimization, to solve a simplified version of this problem. In this paper, we show that ForwardPlanner fails to scale when accounting for all relevant constraints. We describe a novel combination of Squeaky Wheel Optimization (SWO), an incomplete algorithm designed to solve scheduling problems, with previously devised numerical optimization methods and stochastic sampling approaches, as well as heuristics based on reformulations of the SFPP to traditional OR scheduling problems. We show that this new algorithm finds as good or better flight plans as the previous approaches, often with less computation time.

1 Introduction

The Stratospheric Observatory for Infrared Astronomy (SOFIA) is NASA's next generation airborne astronomical observatory. The facility consists of a 747-SP modified to accommodate a 2.5 meter telescope. SOFIA is expected to fly an average of 140 science flights per year over its 20 year lifetime, and will commence operations in early 2005. The SOFIA telescope is mounted aft of the wings on the port side of the aircraft and is articulated through a range of 20° to 60° of elevation. The telescope has minimal lateral flexibility; thus, the aircraft must turn constantly to maintain the telescope's focus on an object during observations. A significant problem in future SOFIA operations is that of scheduling Facility Instrument (FI) flights in support of the SOFIA General Investigator (GI) program, called the SFPP (Single Flight Planning Problem). GIs are expected to propose small numbers of observations, and many observations must be grouped together to make up single flights. Approximately 70 GI flight per year are expected, with 5-15 observations per flight.

* QSS Group, Inc.

Flight planning for the previous generation airborne observatory, the Kuiper Airborne Observatory (KAO), was done by hand; planners had to choose takeoff time, observations to perform, and decide on setup-actions called "dead-legs" to reposition the aircraft. This task frequently required between 6-8 hours to plan one flight [1]. The scope of the flight planning problem for supporting GI observations with the anticipated flight rate for SOFIA makes the manual approach for flight planning daunting. There has been considerable success in automating observation scheduling for ground-based telescopes [1], space-based telescopes such as Hubble Space Telescope [2], Earth Observing Satellites [3] and planetary rovers [4]. However, the SOFIA flight planning problem differs from these problems in a variety of ways. Observations are feasible over large, continuous regions of space and time; observations that can't be done at the current position and time may have an infinite number of setup actions enabling them. The principal feasibility condition for observations is goverened by a nonlinear function over the solution to the equations of motion, complicating the task of finding good heuristics. Temporal constraints are implicit in these continuous constraints; bounding above approximations are hard to calculate and generally weak, making temporal constraint propagation unlikely. Finally, the expense of solving the differential equations impacts the speed of automated planning.

Previously, we developed the first algorithm to solve a simplified version of the SFPP, called ForwardPlanner [5, 6]. ForwardPlanner is a novel combination of AI and OR techniques, including progression planning, lookahead heuristics, biased stochastic sampling, approximations and continuous optimization methods. Initial results with ForwardPlanner on a simplified version of the SFPP were promising; however, we show in this paper that ForwardPlanner fails to scale as more and more constraints (Special Use Airspace (SUAs), runway and airway selection, high-fidelity fuel consumption on takeoff and landing, in-flight altitude changes, calculation of initial fuel load) on valid flight plans were added to the problem description. Computationally expensive lookahead search is needed to obtain good results from ForwardPlanner. Introducing approximations significantly reduces the costs of lookahead, but ultimately leads to poor quality flight plans. Consequently, we seek a new approach to solving the problem.

Squeaky Wheel Optimization (SWO) [7] was originally developed for scheduling problems with an optimization objective. SWO employs a permutation of tasks to schedule, and a fast procedure called a *Constructor* that treats each task in order, ultimately scheduling tasks or rejecting them. The permutation and its resulting schedule are then analyzed by a *Critic* to determine a new permutation that might schedule tasks that were previously rejected. The cycle repeats until all tasks are scheduled or for a fixed number of iterations. SWO was originally evaluated on Graph Coloring [7], and has since been employed for satellite observation scheduling [8] and range scheduling [9], as well as project scheduling with temporal constraints [10]. The promise of SWO for solving the SFPP is that good plans can be found using fewer expensive feasibility checks than ForwardPlanner.

The rest of the paper is organized as follows. We first formally describe the SFPP, the constraints on flight plans, and the optimization criteria used to compare valid flight plans. We then briefly describe the ForwardPlanner and discuss its problems. We then introduce Squeaky Wheel Optimization (SWO) and discuss how to apply it to the SFPP using numerical optimization methods and approximate solutions to OR problems. We show that SWO improves upon ForwardPlanner on a small set of examples. We then discuss a variety of ways to improve the performance of SWO. We describe experiments to validate the approach. Finally, we conclude and discuss future work.

2 SOFIA's Choice

The input to the Single Flight Planning Problem (SFPP) consists of a set of observation requests, each consisting of the Right Ascension (RA) α and Declination (Dec) δ, ob-

[1] Anecdotal evidence provided from conversations with SOFIA staff who worked with KAO.

servation duration, priority; a flight date; maximum fuel load; an altitude profile; earliest takeoff time θ_l and latest landing times θ_u; the designated takeoff and landing airports (which need not be the same); predicted wind and temperature; and a list of SUAs. For a flight plan to be valid, the aircraft must take off from the takeoff airport, land at the landing airport, avoid all SUAs, and consume less than the available fuel at takeoff. The objective is to find a flight plan that maximizes the number of requested observations performed. During flight, *Flight-legs* require tracking an object for a period of time, and are only valid if the object stays within the telescope elevation limits for the requested duration. The observation must also take place in darkness (the sun must be below the horizon). *Dead-legs*, when no observations are performed, can be used to reposition the aircraft to enable flight-legs. A distinguished class of dead-legs are used to take off and return to the landing airport. Since it is intractable to find the best possible plan, we limit ourselves to searching for *good* plans that perform many observations of high priority. Solving the SFPP requires choosing a takeoff time, selecting the set of observations to service, ordering them and inserting necessary dead-legs to ensure that all flight legs are valid.

2.1 Constraints on Valid Flights

In this section we describe the constraints on valid solutions to the SFPP in more detail. The telescope is carried aboard a Boeing 747-SP aircraft. The fuel consumption of each engine depends on the aircraft weight, mach number, outside air temperature, initial altitude and final altitude. The fuel consumption constraints are represented in a lookup table provided by Boeing. The aircraft follows a pre-determined *altitude profile* that describes the maximum permitted altitude at an absolute time after takeoff. Climbs are allowed periodically to decrease fuel consumption. At the end of a leg, if the aircraft is allowed to climb, it climbs to the maximum altitude permitted by the fuel performance table or the altitude profile. The profiles used in this paper were developed assuming standard atmosphere [11]; actual atmospheric conditions and aircraft weight may force the aircraft to fly lower than the altitude profile permits. Predicted wind and temperature are used to calculate the ground track and fuel consumption. Finally, SUAs constrain the ground track of the aircraft by forcing dead-legs to reposition the aircraft. Space precludes describing the fuel consumption constraint in more detail.

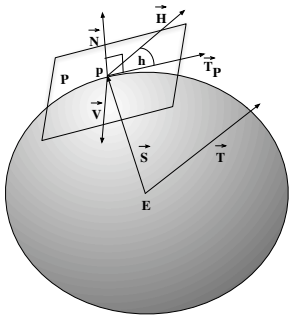

Fig. 1. The Cartesian formulation of the instantaneous equations of motion of the aircraft and the elevation.

The constraints linking aircraft motion and observation feasibility are the most complex and important component of the problem, so we describe them in detail here. SOFIA can view objects between 20° and 60° of elevation (from the plane of flight). If an observation is scheduled, then it must be performed for the requested duration without interruption, and the object must stay within the elevation limits throughout the observation. The elevation of an object depends on the object's coordinates, the aircraft's position and the time.

Checking this constraint requires computing the aircraft's ground track throughout the course of the observation. Figure 1 shows the interaction between the object's coordinates, the aircraft's position, the time, and the telescope elevation. The Earth is modeled as an oblate spheroid $\mathbf{E}$, whose surface is defined by the equation $\frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{c^2} = 1$ where $c < a$. Let $\mathbf{p}$ be the aircraft's current position, (latitude γ and longitude L) and

θ be the (Sidereal) time that the aircraft is at $\mathbf{p}$. Let $\boldsymbol{S}$ be the vector from the center of $\mathbf{E}$ to $\mathbf{p}$. Let $\boldsymbol{T}$ be the vector to an astronomical object o at time θ, and $\mathbf{P}$ as the plane tangent to $\mathbf{E}$ at $\mathbf{p}$. Let $\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}$ be the unit vectors in the x, y, z directions respectively. Let $\boldsymbol{N}$ be the vector normal to $\mathbf{P}$: $\boldsymbol{N} = \frac{p_x}{a^2}\hat{\mathbf{i}} + \frac{p_y}{a^2}\hat{\mathbf{j}} + \frac{p_z}{c^2}\hat{\mathbf{k}}$ (Note that $\boldsymbol{S}$ and $\boldsymbol{N}$ are generally not parallel since $\mathbf{E}$ is a spheroid.) Let $\boldsymbol{T_P}$ be the projection of $\boldsymbol{T}$ onto $\mathbf{P}$; this is the *object azimuth* at $\mathbf{p}$, and is given by $\boldsymbol{T_P} = \boldsymbol{T} - \frac{\boldsymbol{TN}}{||\boldsymbol{N}||^2}\boldsymbol{N}$. Let $\boldsymbol{V}$ be the desired heading of the aircraft. The observatory must track the object inducing $\boldsymbol{T}$, subject to the constraint that the angle between $\boldsymbol{V}$ and $\boldsymbol{T_P}$ is $270°$, because the telescope points out the left-hand side of the aircraft. Let $\mathbf{R}_{\boldsymbol{N}}(\mathbf{270}°)$ be a rotation matrix that rotates a vector $270°$ around $\boldsymbol{N}$, and v be the airspeed of the aircraft; then $\boldsymbol{V} = v\mathbf{R}_{\boldsymbol{N}}(\mathbf{270}°)\frac{\boldsymbol{T_P}}{||\boldsymbol{T_P}||}$. Let $\boldsymbol{H}$ be the elevation vector with respect to $\mathbf{P}$. We also require the angle h between $\boldsymbol{H}$ and $\boldsymbol{T_P}$ obey the constraint $20° \leq h \leq 60°$ throughout an observation. Most targets are sufficiently far from Earth that we can assume $\boldsymbol{H} = \boldsymbol{T} + \boldsymbol{S}$. From vector calculus we then get the equation for the elevation $h = \cos^{-1}\left(\frac{\boldsymbol{HT_P}}{||\boldsymbol{H}||\,||\boldsymbol{T_P}||}\right)$. The angle r between $\boldsymbol{V_d}$ and the object azimuth at the new position $\boldsymbol{T_P}$ is given by: $r = \cos^{-1}\left(\frac{\boldsymbol{V_d T_P}}{||\boldsymbol{V_d}||\,||\boldsymbol{T_P}||}\right)$. Now, $\boldsymbol{T}$ is a function of o and θ; this is because the Earth rotates on its axis. The vector $\boldsymbol{T}$ traces a circle of radius $x^2 + y^2 = \frac{c^2 - d}{c^2}$, where $d = |\frac{\delta}{90°}|$ in 24 hours (see [12] for an explanation of this).

The instantaneous change in $\mathbf{p}$ as the aircraft tracks o is $\frac{d\mathbf{p}}{d\theta} = \boldsymbol{V}$. Since $\boldsymbol{V}$ is a function of $\boldsymbol{T}$, it is a function of $o, \mathbf{p}$ and θ. Solving for the ground track is necessary to compute h over the entire duration of the observation and check the elevation constraints. It is worth noting that this formulation also makes it easy to add the effect of winds by adding the appropriate vectors to $\boldsymbol{V}$, and also correct for aircraft pitch by rotating about $\boldsymbol{V} \times \boldsymbol{N}$, but we omit these for brevity. The ground track and elevation constraints are solved using a specialized 5^{th}-order Runge-Kutta [13] with error-adaptive step sizing.

3 ForwardPlanner and its Discontents

The first fully automated approach to solving the SFPP was ForwardPlanner [5, **?**]. We originally assumed no SUAs and ignored runway and airway selection, ascent and descent, thus simplifying the fuel consumption constraint. ForwardPlanner combines progression based search, continuous numerical optimization, dispatch heuristics and stochastic sampling, resulting in an incomplete randomized algorithm. ForwardPlanner evaluates observations at each phase of a flight, and selected one observation to add to the flight. Suppose the aircraft is at position α, L at time θ after performing some observations. Rather than considering all possible setup actions, ForwardPlanner only considers the *shortest dead-leg* making an observation visible for long enough and allowing the aircraft to subsequently fly to the landing airport. If the shortest dead-leg crosses an SUA, the heading is shifted minimally left or right from the heading of the shortest dead leg until the dead leg misses all SUAs. The duration of the leg is then adjusted to ensure the object is visible for the required duration. If the resulting dead leg is longer than D (an operational limitation on the longest permissible dead-leg), then the observation is rejected. If the flight-leg following this dead-leg crosses any SUA, the observation is rejected. If the observation begins before sunset or ends after sunrise *at the local position*, the observation is rejected. (Remember, changing your position changes the time at which the sun rises or sets.) Finally, if the aircraft cannot return to the landing airport after the observation is performed, the observation is rejected. If the observation survives all of these checks, ForwardPlanner considers it is feasible. Each feasible observation is then evaluated by first adding it to the flight plan, then heuristically adding a fixed number of additional observations. This "lookahead" is performed to estimate the best flight plan possible after adding each observation. These short extensions are evaluated using a weighted sum of the *priority* of the observations performed so far, the *efficiency* (ratio of time spent observing to total flight time) of the (incomplete) flight, the estimated time to return to the designated landing airport, and the total time spent

in turns. The heuristic rank of each observation is treated as the mass of a probability distribution used to select the next observation. Thus, if we have a set of choices C and heuristic values of of these choices $v(c)$, we choose an element $c \in C$ with probability $\frac{v(c)}{\sum_{d \in C} v(d)}$. This technique is similar to Heuristic Biased Stochastic Sampling (HBSS), a technique used for scheduling ground based telescopes [1]. This means that the "best" candidate need not be selected at any stage of the process, but has the highest probability of being selected next. The process of evaluating the feasible observations and adding the next observation to a flight is shown pictorally in Figure 2. ForwardPlanner is a stochastic algorithm, and can be run several times to generate better flights; the ForwardPlanner algorithm sketch is shown in Figure 3.

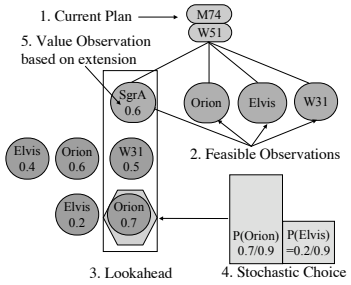

Fig. 2. ForwardPlanner's Evaluate() routine. Each feasible observation in the current plan 1) is added to the plan 2). A fixed number of observations are used to extend the plan 3). Each of these observations is evaluated individually, and the values are used to form a probability distribution; this distribution is sampled 4) to determine how to extend the flight. Once the maximum number of observations in lookahead (2 in this example) is reached, the resulting flight is used to determine how good it is to add the first observation to the current flight 5).

ForwardPlanner()
F is (initially empty) current flight plan
for MaxRepeats
  Select takeoff time
  while not done
    # E is set of feasible observations
    for each unscheduled observation o
      if Feasible(o, p, θ)
        Add p to F; update p, θ
        v=Evaluate(o, F)
        Add (o, v) to E
        Remove o from F
    end **for**
    if E is not empty
      Use values v to select e from E
      Extend F by e; empty E
    else done
  end **for**
return F
end

Fig. 3. A sketch of the ForwardPlanner Algorithm. At each step, all feasible observations are considered as the next observation in the plan. For each feasible observation, the Evaluate() routine builds an extension of the plan to evaluate how good a flight will result. Feasible() is described in Figure 7.

The principal cost of ForwardPlanner is in the lookahead phase, where many legs are constructed to test observation feasibility solely to evaluate an observation, and then are thrown away. Let N be the number of observation requests, let K be the lookahead depth, and let M be the maximum number of observations that can be in any flight plan. Each of *MaxRepeats* loops in ForwardPlanner makes $O(N^2KM)$ calls to Feasible(); a proof of this appears in [5]. It was found empirically that $K = 4$ struck a good balance between computational cost and flight plan quality [5]. The basic algorithm was improved upon in [6] by observing that many expensive dead-leg construction steps in ForwardPlanner could be eliminated by formulating the problem of finding dead-legs to prove feasibility as a zero-finding problem. Suppose an observation is not visible at the current position and time. If we drop the condition on reaching the landing airport, an approximation of the shortest dead leg b, d (b is the heading and d is the duration) has the property that $F_1(b, d) =< f_1(b, d), f_2(b, d) >=< 0, 0 >$ where f_1 is the difference between the object azimuth and the final heading of the aircraft after flying the dead-leg defined by b, d, and f_2 is the difference between the object elevation after flying the dead-leg and the telescope elevation limit closest to the initial object elevation. A similar formulation exists for the shortest dead leg ensuring an observation is visible

throughout a flight leg [6]. We solve for b, d using a Secant Method[2]; the final version of Feasible() used in FowrardPlanner appears in Figure 7. The resulting algorithm is a novel combination of AI progression planning and stochastic sampling and OR numerical optimization techniques for solving a complex constrained optimization problem. This approach reduces the runtime of ForwardPlanner without impacting the value of the flight plans found.

Initial results on solving the simplified version of SFPP with ForwardPlanner were promising [6]. However, adding requirements to avoid SUAs, calculate initial fuel loads in the face of predicted weather, runway and airway selection, and calculating fuel consumption based on altitude changes (especially complex for takeoff and landing). made ForwardPlanner too slow. In particular, SUA evasion and fuel consumption during climb vastly increase the expense of the feasibility check. This is problematic, given that large lookahead and many samples are needed to find good quality plans. Further reductions in runtime can be accomplished by approximately calculating aircraft position after flight legs in the lookahead phase using Euler's Method instead of Runge Kutta. Euler's Method approximates the solution to the ground track by flying a constant heading for a fixed (small) duration relative to the total observation time. The approximation we used does not adequately account for the ellipsoid Earth, wind speed and direction, change of altitude, and estimates fuel consumption based on the last calculated fuel consumption rate. Our intuition was that these approximations would permit a good, fast estimate of the value of inserting an observation. Unfortunately, the heuristic quality degrades too much and leads to poor quality plans. Figures 4 and 5 compare performance on 6 sample problems (we will discuss the SWO results later in the paper). In these experiments, ForwardPlanner was run with MaxRepeats $= 20$. ForwardPlanner without the Euler's Method approximation performs well but takes 8-20 minutes per flight generated. We see that Euler's Method reduces ForwardPlanner's computation time considerably, but leads to plans with fewer scheduled observations in 4 of 6 cases. While some of the cost savings is in lookups to outside air temperature and the fuel table, as well as the switch in integration methods, the vast increase in software complexity required to correct these problems led us to search for new solutions to the problem that allow us to generate good quality flights fast.

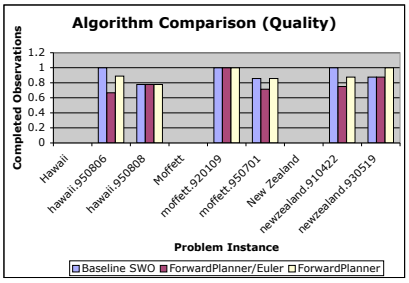

Fig. 4. Comparison of solution quality for ForwardPlanner (with and without Euler's method approximation of flight dynamics) and SWO.

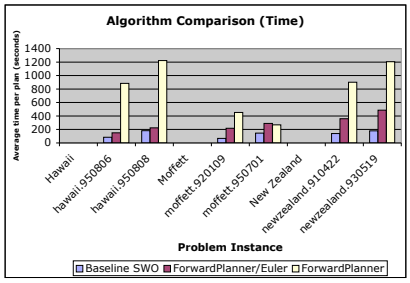

Fig. 5. Comparison of average CPU time for ForwardPlanner (with and without Euler's method approximation of flight dynamics) and SWO.

[2] The previous work incorrectly identified the method used as Newton's Method; since numerical derivatives are used, we actually use a Secant Method, which is the term we will use in this paper.

4 Squeaky Wheel Optimization for the SFPP

SWO employs a permutation of tasks to schedule, and a fast procedure called a *Constructor* that treats each task in order, ultimately scheduling tasks or rejecting them. The permutation and its resulting schedule are then analyzed by a *Critic* to determine a new permutation that might schedule tasks that were previously rejected. The cycle repeats until all tasks are scheduled or for a fixed number of iterations. Figures 6 and 7 describe a *family* of SWO algorithms specialized for solving the SFPP. We discuss the features of this specialized SWO in more detail below.

The constructor assumes that the flight begins at the takeoff time, and that the permutation P imposes a precedence ordering on the observations, and attempts to construct a schedule. If an observation is not trivially visible for the requested duration, the shortest dead-leg is constructed by solving the zero finding problem. If this leg is short enough, SUAs can be avoided, and sufficient fuel remains the observation is added, otherwise it is rejected; this is identical to the procedure used in ForwardPlanner, and is shown in Figure 7. Rejecting the i^{th} observation in P does not imply rejection of $j > i$, so all observations are processed. The best flight B is the flight maximizing $\frac{s}{2} + \frac{e}{2}$, where s is the percentage of requested observations scheduled, and e is the efficiency of the flight (the ratio of time spent observing to flight time) [3]. The final flight plan is checked for SUA violations on the return leg; if there are any, the flight is rejected.

SWO(MaxFlights,MaxRepeats)
F is current flight plan
B is best flight plan
P is a permutation of observations
R is rejected observations
for MaxRepeats
  1. *Generate permutation* P
  for MaxFlights
    2. *Select the takeoff time* θ
    # Construct flight from P
    # p is the current position of F
    for observation $o \in P$
      if Feasible(o, p, θ)
        Add p to F; update p, θ
      else add p to R
    end for
    Update best flight plan B
    if $R = \emptyset$ return F
    else
      3. *Modify* P *by analyzing* F *and* R
  end for
end for
if dead leg home does not violate SUA
  return B

Fig. 6. A sketch of the family of SWO-based Flight Planning Algorithm. Later sections elaborate on options for *1. Generate permutations, 2. Select the takeoff time, and 3. Modify* P *by analyzing* F *and* R.

Feasible(o, p, θ)
o is the observation
p is the current position
D is maximum dead leg duration
(b, d, z) = FindDeadLeg(o, p, θ)
b = heading, d = duration, z = SUA zone
if the dead-leg crosses any SUA zone z
  #Revise dead legs to avoid SUA
  b' is closest heading s.t. all z not crossed
  d' is new duration
  $d = d'; b = b'$
  if $d > D$ **return false**
if observation starts and ends in darkness
  if dead leg home possible following o
    return true
return false

FindDeadLeg(o, p, θ)
#e is the elevation limit o violates at p, θ
Guess dead-leg b, d; calculate r, h after dead-leg
#$f_1(b, d) = r$, $f_2(b, d) = e - h$
while $\langle f_1, f_2 \rangle \neq \langle 0, 0 \rangle$

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial b}(b,d) & \frac{\partial f_1}{\partial d}(b,d) \\ \frac{\partial f_2}{\partial b}(b,d) & \frac{\partial f_2}{\partial d}(b,d) \end{pmatrix} \equiv \begin{pmatrix} p & q \\ r & s \end{pmatrix}$$

  $|J| = ps - qr$
  if $|J| < t$ **then** $|J| = t$ (preserve the sign of $|J|$)
  $db = \frac{qf_2 - sf_1}{|J|}$ and $dd = \frac{pf_1 - rf_2}{|J|}$
  $b = b + db$, $d = d + dd$
  Update r,h

Fig. 7. The feasibility test with the Secant Method for finding dead legs. t, db, dd are tuning parameters. Derivatives are all calculated numerically. r and h are calculated as discussed in Section 2.1.

[3] Efficiency is a secondary criteria for good quality flights.

In order to modify the permutation P, a critic must both select a rejected observation r in R and decide where in P we would like to put r. We want to use the flight plan F built with permutation P to decide how to modify P. Each observation in a flight plan defines a "slot" in which a new observation could be placed. Unlike SWO approaches taken in [9] and [8], we do not perform "blind" migration of jobs in the permutation. Rather, we identify where in the permutation we can move rejected observations to ensure that the resulting schedule is modified. Since we guarantee that a rejected observation will be scheduled during the next construction phase, we run the risk that some later observations in the flight might be displaced. Thus, it is important to estimate how much we "regret" moving an observation to a particular place. Critics must both be fast and produce good quality flights by moving rejected observations without displacing many scheduled observations.

SWO attempts to modify the permutation P by reordering the rejected observations to produce a new flight in which these observations are scheduled, possibly leading to the rejection of other observations. At worst, this might require $O(N^2)$ feasibility checks to determine which slots rejected observations can occupy. While each of *MaxRepeats* calls in ForwardPlanner makes $O(N^2KM)$ flight leg feasibility checks, each such call in SWO makes $O(MaxFlights(N + N^2))$ feasibility checks. As long as *MaxFlights* $< KM$, SWO costs less per invocation than ForwardPlanner; this seems likely since *MaxFlights*, M and K likely scale with N. This makes SWO a good candidate for improving upon ForwardPlanner.

There is a complex interplay between the permutation modification and takeoff time selection. First, it is possible to construct very bad flight plans by poor selection of the takeoff time. Second, the combination of the takeoff time, permutation and the fast scheduler implicitly schedules a subset of the observations. Finally, the fact that permutations are constantly modified allows reconsideration of the takeoff time based on the new permutation. For these reasons, this version of SWO ensures that new takeoff times can be chosen after each modification of the permutation.

4.1 Useful Concepts

In preparation for building our SWO, we introduce some useful concepts.

Time windows during which an object o at Right Ascension α and declination δ is visible at a fixed position can be constructed as follows. If the aircraft is at latitude γ longitude L, the earliest and latest times $\theta_r(o), \theta_s(o)$ at which the observation is visible by SOFIA are given by $\theta_{r,s}(o) = \cos^{-1}\left(\frac{\sin(20)-(\sin\delta)(\sin\gamma)}{(\cos\delta)(\cos\gamma)}\right)+L+\alpha$ [12]. The $\sin(20)$ term arises from the fact that SOFIA's lower elevation limit is 20°. Note that $\cos^{-1}(x)$ has 2 solutions, which provide the earliest rise time $\theta_r(o)$ and latest set time $\theta_s(o)$ of the object at this position. The time of sunset and sunrise at this position can be used to further tighten this window. There can be at most 2 feasible windows since all objects period is 24 (sidereal) hours and the aircraft stays aloft less than 10 hours. For example, an object can rise above the maximum elevation limit, then drop back into view. In our critics, by default we use the *first* feasible window. We will also use the time at which an object reaches its maximum elevation (above the local horizon), called the *transit time*. This is simply $\frac{\theta_s(o)+\theta_r(o)}{2}$.

The SFPP can be relaxed by approximating time windows for observations as described in the previous paragraph, effectively pretending that the observatory is fixed at some location. This leaves a problem in which observations have release times (earliest rise times), due dates (latest set times), occupy a unary resource (the telescope). This approximation is not bounding, because objects may rise earlier and set later at different positions than the one used to calculate the time windows. Since SOFIA has a maximum and maximum telescope elevation limit, the true feasibility windows of objects may not be convex. Additionally, objects could set then rise during the night, but usually objects are observed at times of year when they are visible all night (and thus achieve their maximum elevation sometime during the night). The resulting problem

is $1|r_i; p_i; d_i| \sum w_i U_i$ according to Graham's hierarchy, a well-studied problem in AI and OR which Karp proved $\mathcal{NP}$-complete [14]. We use approximate solutions of this problem in our takeoff-time selection method.

4.2 Generating Initial Permutations

We considered the following ways of generating the initial permutation:

Random selection **Uniform**: If there are N observations, one of the $N!$ permutations is chosen uniformly at random.

Sort by Earliest Start Time **Rise** at the takeoff airport: We calculate $\theta_r(o)$ as described in the previous section. The intuition behind this ordering is that flights often occupy the whole night, so beginning observations as early as possible is a good initial guess. Furthermore, this allows the largest time window to observe any object.

Sort by Latest Start Time **Set** at the takeoff airport: We calculate $\theta_s(o)$ as described in the previous section. Observing an object as late as possible may be a cheap method of ensuring enough time remains to schedule necessary dead-legs.

Sort by Transit Time **Transit** at the (landing) airport: The intuition here is that this allows observing very nearby the airport; while one object is being observed, the next object moves closer to the landing airport, allowing the aircraft to "loiter" nearby.

4.3 Generating Takeoff Time

As we previously observed, due to the complex nature of the visibility constraints, choosing a good takeoff time is important to constructing good flight plans. We therefore break down the takeoff time generation process into 2 phases: determining a *range* of takeoff times, and *sampling* from the range. We considered several takeoff time methods:

Estimated flight duration **FlightDur**: If we simply assume that the aircraft will stay aloft as long as possible, we can estimate the flight duration f from the initial fuel load and flight profile. The takeoff time range is $[\theta_l, \theta_u - f]$. Since this quantity is independent of the permutation, it needs be calculated only once. However, especially in the summertime for long flights, f will exceed the duration of the night. This approach will overestimate the number of observations that can actually be performed, and reduce the takeoff time range to one time (roughly half an hour before sunset).

Minimum of Earliest Start Times **Min Rise**: We can calculate the minimum over all $\theta_r(o)$ at the takeoff airport, and "pad" this by the amount of time needed to climb to operational altitude. Since this quantity is independent of the permutation, it needs be calculated only once. Only one takeoff time is generated by this approach.

Optimize **First-Observation** in Permutation: It is clear that $\theta_r(o)$ can be a bounding above approximation to the earliest time when an observation can be performed; to see why, observe that flying towards the observation makes it possible to observe it earlier. Another approach is to assume that the first observation in a permutation is meant to be observed, and to calculate the earliest time at which this observation can be performed. Binary search over takeoff times is performed to find the takeoff time leading to the earliest feasible observation time for the first observation. Only one feasible takeoff time is generated by this approach. As the first observation in the permutation can change, the takeoff time will need to be recalculated.

Approximate solution to the relaxed scheduling problem **Feas-Sched**: We use the $\theta_r(o)$ and $\theta_s(o)$ calculated at the takeoff airport to approximate the time windows for the observations and induce the relaxed scheduling problem $1|r_i; p_i; d_i| \sum w_i U_i$. Since all we are interested in is a range of takeoff times, we consider only fast approximation algorithms. A feasible solution to the relaxed scheduling problem can be generated using the permutation as an ordering heuristic, and either greedily scheduling from the beginning or the end of the permutation. It is trivial to see that different feasible schedules, and different takeoff time ranges, can be generated by scheduling forwards or backwards; this leads to two methods, **Feas-Sched (Fwd)** and **Feas-Sched (Bkwd)**. Once

a feasible solution is generated, we calculate the slack of the first feasible observation, again "padding" for the time to climb to altitude.

If a range of takeoff times is generated, we select from them uniformly at random.

4.4 Modifying the Permutation with Critics

In what follows, assume the problem instance contains N observation requests. All of our critics use the biased sampling approach described earlier to make selections. Recall that if we have a set of choices C and values of of these choices $v(c) \in C$, we choose an element $c \in C$ with probability $\frac{v(c)}{\sum_{d \in C} v(d)}$.

We explored the following five critics to modify the permutation:

1-Phase: We first determine for each rejected observation o whether it is feasible in each slot s. This test uses the feasibility test in Figure 7 assuming the aircraft begins at the position and time at the beginning of slot s. For each feasible pair (o, s) we examine the time at which the new observation o ends. Since the new observation is guaranteed to be feasible, successive observations will be delayed, both due to the duration of the new observation and its dead leg (if any). We then evaluate the rate of change of the elevation of each successive observation to find out if it would still be visible at the same position at the later time. This is obviously an approximation, since the aircraft position would change after the newly inserted observation. Furthermore, it doesn't consider the possibility that unscheduled observations in the permutation could be added, so it is a conservative regret estimate. Let X be the set of observations we estimate are made infeasible by o. We then calculate $v(o, s)$ for the sampling probabilities as follows. If s is the first or last slot or one for which $X = \emptyset$, then $v(o, s) = N$. Otherwise, $v(o, s) = \left(\sum_{x \in X} u(x)\right)^{-1}$, where $u(x) = 0.5$ if x had a dead-leg before it, and $u(x) = 1$ if not. This penalizes choices that incur more regret, with the assumption that replaced observations with dead-legs provide more flexibility for other observations to be added later.

Obs-Slot: We first determine for each rejected observation o whether it is feasible in each slot s. We then randomly choose a feasible observation o from those that could go into some slot s. We calculate sampling probabilities as follows: if an observation is visible in s slots, the heuristic is $v(o) = N + 1 - s$. (Observations visible nowhere are not chosen.) We then calculate $v(o, s)$ as described above for those s in which o is feasible, and randomly choose the slot for o.

Slot-Obs: We first determine for each rejected observation o whether it is feasible in each slot s. We then randomly choose a slot in which at least one rejected observation is feasible. We calculate sampling probabilities as follows: if v observations are visible in a slot, and the problem instance contains N observations the heuristic is $v(s) = N + 1 - v$. We then calculate $v(o, s)$ as described above for those o feasible in s, and choose randomly the observation to move to s.

Time$_t$: For this critic, we use $\theta_s(o)$ and $\theta_r(o)$ at the takeoff airport, which can be calculated once and needs never be repeated. The critic first chooses a feasible observation o. We calculate sampling probabilities as follows: $v(o) = \frac{1}{\theta_s(o) - \theta_r(o)}$. We then determine which slots s are feasible for o, and calculate $v(o, s)$ as described above. Finally, we randomly choose the observation to move to s using $v(o, s)$.

Time$_f$: Calculating $\theta_r(o), \theta_s(o)$ at the takeoff airport is clearly inaccurate. We can instead calculate $\theta_s(o)$ and $\theta_r(o)$ at each slot in the current flight, but at a higher computational cost. We calculate sampling probabilities as follows: if S is the set of slots in the flight, $v(o) = \min_{s \in S} \frac{1}{(\theta_s(o) - \theta_r(o))_s}$. We then choose o according to $v(o)$. We then determine which slots s are feasible for o, and calculate $v(o, s)$ as described above. Finally, we randomly choose the observation to move to s using $v(o, s)$.

As a final wrinkle, we can modify the permutation by moving k rejected objects rather than just one. The idea here is that multiple rejected observations could be reordered *independently* and potentially improve the flight plan using fewer construction steps. This idea was successfully employed by [8] and [9] to speed up SWO.

5 Identifying the Right SWO Features

Our approach to finding the best SWO features is to begin with a baseline algorithm: **Flight-Duration** based takeoff time range selection, **Uniform** random initial permutation, and the **Time**$_t$ critic. We will use the Wilcoxon Signed Ranked Test [15] to determine whether using one feature is superior (finds better quality flights) to the baseline SWO; we will select a small subset of promising algorithms to generate the next algorithm. In the presentation of the Wilcoxon test results, X indicates the tests leading to different values, positive z indicates an algorithm variant is likely to perform better than the baseline, while a negative z indicates an algorithm variant is likely to perform worse than baseline. Criticality measurements are typically given in ranges; criticalities of > 0.05 are not considered statistically significant.

6 Empirical Results

In this section we present empirical results for varying facets of SWO in order to find the best overall algorithm for solving the SFPP.

6.1 Sample Problems

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Airport	H	H	H	H	M	M	M	M	M	M	M	M	M	M	M	M
Date	8/6	8/8	8/10	8/12	1/9	1/10	1/16	6/16	6/18	6/19	6/30	7/6	8/12	8/16	4/4	4/5
# Obs	9	9	10	10	7	8	8	6	10	8	8	6	11	10	9	9
Index	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Airport M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
Date	4/6	4/11	4/12	4/14	4/19	5/4	5/8	7/1	7/6	8/2	8/22	8/24	8/26	8/29	9/1	9/19
# Obs	10	8	8	8	10	10	6	7	4	6	9	8	11	10	8	7
Index	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	
Airport	M	M	M	M	M	M	M	MH	MH	MH	N	N	N	N	N	
Date	9/20	9/21	9/23	9/26	9/28	9/29	10/4	6/21	7/12	8/4	11/25	4/22	5/11	5/15	5/19	
Obs	7	3	10	8	8	8	4	8	7	7	10	8	8	8	8	

Fig. 8. Characteristics of Single Day Instances.

We used as a benchmark flights previously flown on KAO, described in [5] to determine the utility of our new techniques. In Figure 8 we tabulate the number of observations, he archived flight duration, and the airport. Flights from Moffett Field, CA are denoted with an M; flights originating in Moffett and ending in Hawaii are denoted MH; flights from Hawaii are denoted H, and flights from New Zealand are denoted N. Takeoff time is between sunset and sunrise (calculated for each day and year of flight). Wind and temperature data from European Center for Medium Range Weather Forecasting. [4] are used to calculate ground tracks and fuel consumption. The initial fuel load is also calculated for each flight, and is based on the altitude profile 4 from [11]. This profile conforms to realistic expectations that good observing will require an altitude of at least 39000 ft. Finally, SUAs impact flights from Moffett and Hawaii; we use data from the National Geospatial Intelligence Agency's Digital Aeronautical Flight Information File.

The priorities of all observations are identical, and all observations could be scheduled for the KAO flights. While SOFIA's performance characteristics differ from KAO and its elevation limits are different, we found ForwardPlanner was able to schedule all observations for most of the tests we constructed [6]. Thus, the principal goal is to

[4] www.ecmwf.int

find an efficient flight with all of the observations scheduled. The maximum dead-leg duration D was set to 4 hours. For the dead-leg search using Secant Method we used a step cutoff of 150 and error tolerance $t = 10^{-6}$. The step parameters used in forward differencing were: $s_1 = 0.01^\circ$ and $s_2 = 60$ seconds. When CPU times are reported, these experiments were run on a Sun Workstation with dual 600 MHz CPUs and 2048 Mb memory. Unless otherwise stated, *MaxFlights* = 20 and *MaxRepeats* = 10.

6.2 Choosing Takeoff Times

The results of varying the takeoff time selection while holding all other aspects of SWO the same are shown in Figure 9. In this figure we present the Wilcoxon Ranked Sign test output for the best percentage of the observations found by SWO. Recall that we compare each new SWO variant to the baseline SWO described in the previous section according to the quality of the flights. In what follows, our "best" SWO variants are those "most likely to exceed the quality of the baseline SWO".

Takeoff Range	X	Z	Crit.
Min Rise	17	-2.218	[0.01,0.025]
First Observation	18	-2.057	[0.01,0.025]
Feas-Sched (Fwd)	12	1.313	>0.05
Feas-Sched (Bkwd)	16	1.822	[0.025,0.05]

Fig. 9. Wilcoxon Ranked Sign Test results comparing SWO Takeoff Time variants to SWO Baseline.

Backwards scheduling to produce a relaxed feasible schedule **Feas-Sched(Bkwd)** did best. The least "informed" approach, **Min Rise**, performs worst. Optimizing the takeoff time range of the first observation also did not perform well. Both of these approaches perform worse than the baseline SWO, which uses **Flight-Duration**.

Curiously, **Feas-Sched(Fwd)** did not perform as well as **Feas-Sched(Bkwd)**. It is possible that scheduling backwards produces a larger takeoff time range, thereby increasing flexibility, but more work is needed to understand this result. Also, as we will see later, **Feas-Sched(Fwd)** takes more CPU time.

6.3 Generating Initial Permutations

For this series of tests, we tested the **Feas-Sched (Bkwd)** variant of takeoff time selection with the different initial permutation methods. The results of varying the permutation selection while using the baseline critic are shown in Figure 10. Notice that **Uniform** is our baseline permutation method, and thus the first line of Figure 10 is repeated from table 9.

Permutation	X	Z	Crit.
Uniform	16	1.822	[0.025,0.05]
Rise	14	2.055	[0.01,0.025]
Set	14	1.428	> 0.05
Transit	14	1.490	> 0.05

Fig. 10. Wilcoxon Ranked Sign Test results comparing SWO Initial Permutation ordering variants to SWO Baseline.

Previous work indicates that "informed" initial permutations improve the performance of SWO when compared to random permutations. We find this to be the case as well; **Rise** coupled with the **Feas-Sched (Bkwd)** performs best when compared to the baseline SWO. Surprisingly, **Uniform** performs second best, but is not as good as **Rise**. We have no intuition for why **Set** and **Transit** are worse than **Uniform**.

6.4 Modifying Permutations

For this series of tests, we tested the **Feas-Sched (Bkwd)** takeoff time generation method and **Rise** initial permutation generation method with each critic method. In each case, only one rejected observation was moved per critic application. The results

of varying the critics are shown in Figure 11. Notice that **Time**$_t$ is our baseline critic method, and thus the first line of Figure 11 is repeated from table 10.

As expected, **1-Phase** is quite good. Also as expected, we see that **Time**$_t$ is not as good as **Time**$_f$. Somewhat surprisingly, though, **Time**$_t$ and **Time**$_f$ are superior to **Obs-Feas** and **Slot-Feas**, even though the former do not correctly identify the feasible observation-slot combinations, while the latter do not. This suggests that even crude estimates of time are important when building the critics, and demonstrates that simply using slot counts is not good enough.

Our final critic experiments use **Feas-Sched (Bkwd)** takeoff time generation, **Rise** based initial permutation selection, and **1-Phase** critic. In this experiment we vary the number of rejected observations that are moved. The regret values are still used to sample, and are renormalized between samples. The number of observations is moderately low, so we limited ourselves to experiments moving $2, 3$ or all rejected observations. As we see, we don't always benefit from increasing the number of rejected observations that are moved; moving 2or 3 rejects isn't as much of an improvement as moving 1, but moving all rejects is clearly better than moving 1.

Critic	X	Z	Crit.
Time$_t$	14	2.055	[0.01,0.025]
Time$_f$	16	2.210	[0.01 0.025]
Obs-Feas	14	1.710	[0.025 0.05]
Slot-Feas	16	1.641	$>$0.05
1-Phase	14	2.338	[0.005 0.01]

Fig. 11. Wilcoxon Ranked Sign Test results comparing SWO Critic variants to SWO Baseline.

Rejects	N	Z	Crit.
1	14	2.338	[0.005, 0.01]
2	13	2.148	[0.01, 0.025]
3	13	2.253	[0.01, 0.025]
all	15	2.541	≈ 0.005

Fig. 12. Wilcoxon Ranked Sign Test results comparing critics moving variable numbers of rejected observations to SWO Baseline.

6.5 The Best Algorithms

First, we revisit Figures 4 and 5. The baseline SWO generates plans of as good or better quality as ForwardPlanner. It runs at a fraction of the time of ForwardPlanner without the Euler's Method approximation speedup, and often is faster than ForwardPlanner with Euler's Method. The results show that, for these 6 problems, the baseline SWO is capable of producing quality plans.

We next compare the CPU performance of the SWO algorithms. In order to make sense of this analysis, it is important to note that SWO terminates if all observations are scheduled. We compare algorithm performance in Figure 13 using the mean and standard deviation in CPU times for all 20 runs of the different algorithms; CPU times are given in seconds. We also reproduce the Wilcoxon signed rank test results comparing the quality of the flights of each SWO version to the SWO baseline. Overall, adding features that further improve the quality of flights leads to roughly a factor of two increase in CPU time. The takeoff time selection method imposes a significant computational burden on SWO, as can be seen by the increase in the mean CPU time. While the critics also impose a computational burden on SWO, we actually see a *reduction* in CPU time compared to those methods without the intelligent critics; this is likely due to the early termination of SWO when all observations are scheduled.

Analyzing the CPU time on a case by case basis, we find that our worst-case performance hit is roughly a factor of 10 increase in CPU time between the baseline SWO and the best SWO, which is moderately high. However, the vast majority of the time the CPU time hit is under a factor of 2. The resulting SWO algorithms deliver significantly better quality flights than ForwardPlanner with Euler's Approximation, at roughly comparable run times.

Name	Baseline	T/O	Perm.	Critic	Swaps
Takeoff Range	**FlightDur**	**Feas-Sched (Fwd)**	⇒	⇒	⇒
Permutation	**Uniform**	⇒	**Rise**	⇒	⇒
Critic	**Rise**$_t$	⇒	⇒	**1-Phase**	⇒
Swaps	1	1	1	1	all
Mean	63.728	113.071	187.612	166.486	145.501
Sdev	29.976	77.623	144.755	108.985	86.427
X	-	16	14	14	15
Z	-	1.823	2.055	2.338	2.541
Crit	-	[0.025,0.05]	[0.01,0.025]	[0.005,0.01]	≈ 0.005

Fig. 13. Comparison of mean and variance of SWO CPU times for all "incremental best" SWO variants identifying best SWO features.

7 Conclusions and Future Work

We described the SFPP, a difficult mixed discrete and continuous constrained optimization problem. We describe ForwardPlanner, an initially promising approach mixing techniques from AI and OR, which ultimately fails to scale for the SFPP. We have described the application of SWO to the SFPP problem. As with our previous approach, ForwardPlanner, the resulting algorithm combines AI and OR techniques to solve a difficult constrained optimization scheduling problem. Our results indicate that SWO is a powerful technique that delivers higher quality flight plans in less time than ForwardPlanner, our previous approach to the SFPP. The quality of flights found by SWO can be increased even further, at a reasonably increase in CPU time.

The SFPP differs from many traditional OR problems in that the temporal and resource constraints are implicit functions of the constraints on elevation and the constraints of motion of the aircraft. The combination of relaxations and continuous optimization method used in ForwardPlanner to reduce the infinite space of setup actions lead to an efficient constructor for our SWO algorithm. We also show that relaxations of the SFPP lead to traditional OR problems, and employ heuristic solutions to these problems in our SWO approach to good effect. In particular, the takeoff time selection method based on greedy solutions to $1|r_i; d_i|\sum w_i U_I$ proved to be an important component of the best quality SWO algorithm. Finally, we verify two conclusions from previous work in SWO. First, informed permutation construction techniques improve SWO performance over random permutation generation. Second, swapping many rejected observations per critic application pays off well in terms of both the quality of solutions and speed of SWO. These lessons may serve others working on complex constrained optimization problems with mixes of discrete and continuous variables.

There is considerably work left to do on the SFPP. Our experiments assumed all observations were of equal value; it is easy to generalize our SWO to handle variable priority, but empirical studies are needed to ensure SWO finds high quality flights. Our benchmark included problems for which it was always possible to schedule all observations. SWO can be modified for problems where this is impossible. Ongoing work shows SWO works well even when this is not the case; again, further tests are required to ensure good performance. In particular, CPU times will likely increase when early termination is no longer likely. Additionally, for each observation, minimizing average line-of-sight water vapor is an important objective. Initial results with SWO show promise, but more work is needed. Finally, the SFPP also requires that we build series of flights rather than just a single flight. Preliminary flight series testing indicates that SWO is a promising technique for building flight series, but the basic algorithm requires some modifications to ensure good performance.

8 Acknowledgments

We would like to thank European Center for Medium Range Weather Forecasting for the use of the climatology data, Michael A. K. Gross for his ongoing assistance in this project, and Tien Ba Dinh for prototyping SWO for the SFPP. This work was funded by the SOFIA Projects Office and by the NASA Intelligent Systems Program.

References

1. Bresina, J.: Heuristic-biased stochastic sampling. In: Proceedings of the 13th National Conference on Artificial Intelligence. (1996)
2. Johnston, M., Miller, G.: Spike: Intelligent scheduling of the hubble space telescope. In Zweben, M., Fox, M., eds.: Intelligent Scheduling. Morgan Kaufmann Publishers (1994)
3. Potter, W., Gasch, J.: A photo album of earth: Scheduling landsat 7 mission daily activities. In: Proceedings of the International Symposium Space Mission Operations and Ground Data Systems. (1998)
4. Smith, D.: Choosing objectives in over-subscription planning. Proceedings of the 14^{th} International Conference on Automated Planning and Scheduling (2004)
5. Frank, J., Kürklü, E.: Sofia's choice: Scheduling observations for an airborne observatory. In: Proceedings of the 13^{th} International Conference on Automated Planning and Scheduling. (2003)
6. Frank, J., Gross, M.A.K., Kürklü, E.: Sofia's choice: An ai approach to scheduling airborne astronomy observations. In: Proceedings of the 16^{th} Conference on Innovative Applications of Artificial Intelligence. (2004)
7. Joslin, D., Clements, D.: Squeaky wheel optimization. Journal of Artificial Intelligence Research **10** (1999) 353 – 373
8. Globus, A., Crawford, J., Lohn, J., Pryor, A.: A comparison of techniques for scheduling earth observing satellites. In: Proceedings of the 16^{th} Conference on the Innovative Applications of Artificial Intelligence. (2004)
9. Barbalescu, L., Whitley, D., Howe, A.: Leap before you look: An effective strategy in an oversubscribed scheduling problem. In: Proceedings of the 19^{th} National Conference on Artificial Intelligence. (2004)
10. Smith, T., Pyle, J.: An effective algorithm for project scheduling with arbitrary temporal constraints. In: Proceedings of the 19^{th} National Conference on Artificial Intelligence. (2004)
11. Becklin, E., Horn, J.: High-latitude observations on sofia. Publications of the Astronomical Society of the Pacific **113** (2001)
12. Meeus, J.: Astronomical Algorithms. Willmann-Bell, Inc. (1991)
13. Cash, J.R., Karp, A.H.: A variable order runge-kutta method for initial value problems with rapidly varying right hand sides. ACM Transactions on Mathematical Software **16** (1990) 201–222
14. Brücker, P.: Scheduling Algorithms. Springer (1998)
15. Lindgren, B.: Statistical Theory. Macmillian (1976)